The semantics of grammar formalisms seen as computer languages

The Semantics of Grammar Formalisms Seen as Computer Languages

Fernando C. N. Pereira and Stuart M. Shieber
Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

Abstract

The design, implementation, and use of grammar formalisms for natural language have constituted a major branch of computational linguistics throughout its development. By viewing grammar formalisms as just a special case of computer languages, we can take advantage of the machinery of denotational semantics to provide a precise specification of their meaning. Using Dana Scott's domain theory, we elucidate the nature of the feature systems used in augmented phrase-structure grammar formalisms, in particular those of recent versions of generalized phrase structure grammar, lexical functional grammar and PATR-II, and provide a denotational semantics for a simple grammar formalism. We find that the mathematical structures developed for this purpose contain an operation of feature generalization, not available in those grammar formalisms, that can be used to give a partial account of the effect of coordination on syntactic features.

1. Introduction[1]

The design, implementation, and use of grammar formalisms for natural language have constituted a major branch of computational linguistics throughout its development. However, notwithstanding the obvious superficial similarity between designing a grammar formalism and designing a programming language, the design techniques used for grammar formalisms have almost always fallen short with respect to those now available for programming language design.

Formal and computational linguists most often explain the effect of a grammar formalism construct either by example or through its actual operation in a particular implementation. Such practices are frowned upon by most programming-language designers; they become even more dubious if one considers that most grammar formalisms in use are based either on a context-free skeleton with augmentations or on some closely related device (such as ATNs), consequently making them obvious candidates for a declarative *semantics*[2] extended in the natural way from the declarative semantics of context-free grammars.

The last point deserves amplification. Context-free grammars possess an obvious declarative semantics in which nonterminals represent sets of strings and rules represent n-ary relations over strings. This is brought out by the reinterpretation familiar from formal language theory of context-free grammars as polynomials over concatenation and set union. The grammar formalisms developed from the definite-clause subset of first order logic are the only others used in natural-language analysis that have been accorded a rigorous declarative semantics—in this case derived from the declarative semantics of logic programs [3,12,11].

Much confusion, wasted effort, and dissension have resulted from this state of affairs. In the absence of a rigorous semantics for a given grammar formalism, the user, critic, or implementer of the formalism risks misunderstanding the intended interpretation of a construct, and is in a poor position to compare it to alternatives. Likewise, the inventor of a new formalism can never be sure of how it compares with existing ones. As an example of these difficulties, two simple changes in the implementation of the ATN formalism, the addition of a well-formed substring table and the use of a bottom-up parsing strategy, required a rather subtle and unanticipated reinterpretation of the register-testing and -setting actions, thereby imparting a different meaning to grammars that had been developed for initial top-down backtrack implementation [22].

Rigorous definitions of grammar formalisms can and should be made available. Looking at grammar formalisms as just a special case of computer languages, we can take advantage of the machinery of *denotational semantics* [20] to provide a precise specification of their meaning. This approach can elucidate the structure of the data objects manipulated by a formalism and the mathematical relationships among various formalisms, suggest new possibilities for linguistic analysis (the subject matter of the formalisms), and establish connections between grammar formalisms and such other fields of research as programming-

[1]The research reported in this paper has been made possible by a gift from the System Development Foundation.

[2]This use of the term "semantics" should not be confused with the more common usage denoting that portion of a grammar concerned with the meaning of object sentences. Here we are concerned with the meaning of the metalanguage.

language design and theories of abstract data types. This last point is particularly interesting because it opens up several possibilities—among them that of imposing a *type discipline* on the use of a formalism, with all the attendant advantages of compile-time error checking, modularity, and optimized compilation techniques for grammar rules, and that of relating grammar formalisms to other knowledge representation languages [1].

As a specific contribution of this study, we elucidate the nature of the feature systems used in augmented phrase-structure grammar formalisms, in particular those of recent versions of generalized phrase structure grammar (GPSG) [5,15], lexical functional grammar (LFG) [2] and PATR-II [18,17]; we find that the mathematical structures developed for this purpose contain an operation of *feature generalization*, not available in those grammar formalisms, that can be used to give a partial account of the effect of coordination on syntactic features.

Just as studies in the semantics of programming languages start by giving semantics for simple languages, so we will start with simple grammar formalisms that capture the essence of the method without an excess of obscuring detail. The present enterprise should be contrasted with studies of the generative capacity of formalisms using the techniques of formal language theory. First, a precise definition of the semantics of a formalism is a prerequisite for such generative-capacity studies, and this is precisely what we are trying to provide. Second, generative capacity is a very coarse gauge: in particular, it does not distinguish among different formalisms with the same generative capacity that may, however, have very different semantic accounts. Finally, the tools of formal language theory are inadequate to describe at a sufficiently abstract level formalisms that are based on the simultaneous solution of sets of constraints [9,10]. An abstract analysis of those formalisms requires a notion of partial information that is precisely captured by the constructs of denotational semantics.

2. Denotational Semantics

In broad terms, denotational semantics is the study of the connection between programs and mathematical entities that represent their input-output relations. For such an account to be useful, it must be *compositional*, in the sense that the meaning of a program is developed from the meanings of its parts by a fixed set of mathematical operations that correspond directly to the ways in which the parts participate in the whole.

For the purposes of the present work, denotational semantics will mean the semantic domain theory initiated by Scott and Strachey [20]. In accordance with this approach, the meanings of programming language constructs are certain partial mappings between objects that represent partially specified data objects or partially defined states of computation. The essential idea is that the meaning of a construct describes what information it adds to a partial description of a data object or of a state of computation. Partial descriptions are used because computations in general may not terminate and may therefore never produce a fully defined output, although each individual step may be adding more and more information to a partial description of the undeliverable output.

Domain theory is a mathematical theory of considerable complexity. Potential nontermination and the use of functions as "first-class citizens" in computer languages account for a substantial fraction of that complexity. If, as is the case in the present work, neither of those two aspects comes into play, one may be justified in asking why such a complex apparatus is used. Indeed, both the semantics of context-free grammars mentioned earlier and the semantics of logic grammars in general can be formulated using elementary set theory [7,21].

However, using the more complex machinery may be beneficial for the following reasons:

- *Inherent partiality*: many grammar formalisms operate in terms of constraints between elements that do not fully specify all the possible features of an element.
- *Technical economy*: results that require laborious constructions without utilizing domain theory can be reached trivially by using standard results of the theory.
- *Suggestiveness*: domain theory brings with it a rich mathematical structure that suggests useful operations one might add to a grammar formalism.
- *Extensibility*: unlike a domain-theoretic account, a specialized semantic account, say in terms of sets, may not be easily extended as new constructs are added to the formalism.

3. The Domain of Feature Structures

We will start with an abstract denotational description of a simple feature system which bears a close resemblance to the feature systems of GPSG, LFG and PATR-II, although this similarity, because of its abstractness, may not be apparent at first glance. Such feature systems tend to use data structures or mathematical objects that are more or less isomorphic to directed graphs of one sort or another, or, as they are sometimes described, partial functions. Just what the relation is between these two ways of viewing things will be explained later. In general, these graph structures are used to encode linguistic information in the form of attribute-value pairs. Most importantly, partial information is critical to the use of such systems—for instance, in the variables of definite clause grammars [12] and in the GPSG analysis of coordination [15]. That is, the elements of the feature systems, called *feature structures* (alternatively, feature bundles, f-structures [2], or terms) can be partial in some sense. The partial descriptions, being in a domain of attributes and complex values, tend to be equational in nature: some feature's value is equated with some other value. Partial descriptions can be understood

in one of two ways: either the descriptions represent sets of fully specified elements of an underlying domain or they are regarded as participating in a relationship of partiality with respect to each other. We will hold to the latter view here.

What are feature structures from this perspective? They are repositories of information about linguistic entities. In domain-theoretic terms, the underlying domain of feature structures F is a recursive domain of partial functions from a set of *labels* L (features, attribute names, attributes) to complex values or primitive atomic values taken from a set C of *constants*. Expressed formally, we have the domain equation

$$F = [L \rightharpoonup F] + C \quad .$$

The solution of this domain equation can be understood as a set of trees (finite or infinite) with branches labeled by elements of L, and with other trees or constants as nodes. The branches $l_1, \ldots, l_m$ from a node n point to the values $n(l_1), \ldots, n(l_m)$ for which the node, as a partial function, is defined.

4. The Domain of Descriptions

What the grammar formalism does is to talk *about* F, not *in* F. That is, the grammar formalism uses a domain of descriptions of elements of F. From an intuitive standpoint, this is because, for any given phrase, we may know facts about it that cannot be encoded in the partial function associated with it.

A partial description of an element n of F will be a set of equations that constrain the values of n on certain labels. In general, to describe an element $x \in F$ we have equations of the following forms:

$$\begin{aligned} (\cdots(x(l_{i_1}))\cdots)(l_{i_m}) &= (\cdots(x(l_{j_1}))\cdots)(l_{j_n}) \\ (\cdots(x(l_{i_1}))\cdots)(l_{i_m}) &= c_k \quad , \end{aligned}$$

which we prefer to write as

$$\begin{aligned} \langle l_{i_1} \cdots l_{i_m} \rangle &= \langle l_{j_1} \cdots l_{j_m} \rangle \\ \langle l_{i_1} \cdots l_{i_m} \rangle &= c_k \end{aligned}$$

with x implicit. The terms of such equations are constants $c \in C$ or *paths* $\langle l_{i_1} \cdots l_{i_m} \rangle$, which we identify in what follows with strings in L^*. Taken together, constants and paths comprise the *descriptors*.

Using Scott's *information systems* approach to domain construction [16], we can now build directly a characterization of feature structures in terms of information-bearing elements, equations, that engender a system complete with notions of compatibility and partiality of information.

The information system **D** describing the elements of F is defined, following Scott, as the tuple

$$\mathbf{D} = \langle D, \Delta, \mathrm{Con}, \vdash \rangle \quad ,$$

where D is a set of *propositions*, Con is a set of finite subsets of D, the *consistent* subsets, $\vdash$ is an *entailment* relation between elements of Con and elements of D and Δ is a special *least informative element* that gives no information at all. We say that a subset S of D is *deductively closed* if every proposition entailed by a consistent subset of S is in S. The *deductive closure* $\overline{S}$ of $S \subseteq D$ is the smallest deductively closed subset of D that contains S.

The descriptor equations discussed earlier are the propositions of the information system for feature structure descriptions. Equations express constraints among feature values in a feature structure and the entailment relation encodes the reflexivity, symmetry, transitivity and substitutivity of equality. More precisely, we say that a finite set of equations E entails an equation e if

- *Membership*: $e \in E$
- *Reflexivity*: e is Δ or $d = d$ for some descriptor d
- *Symmetry*: e is $d_1 = d_2$ and $d_2 = d_1$ is in E
- *Transitivity*: e is $d_1 = d_2$ and there is a descriptor d such that $d_1 = d$ and $d = d_2$ are in E
- *Substitutivity*: e is $d_1 = p_1 \cdot d_2$ and both $p_1 = p_2$ and $d_1 = p_2 \cdot d_2$ are in E
- *Iteration*: there is $E' \subset E$ such that $E' \vdash e$ and for all $e' \in E'$ $E \vdash e'$

With this notion of entailment, the most natural definition of the set Con is that a finite subset E of D is consistent if and only if it does not entail an *inconsistent equation*, which has the form $c_1 = c_2$, with c_1 and c_2 as distinct constants.

An arbitrary subset of D is consistent if and only if all its finite subsets are consistent in the way defined above. The consistent and deductively closed subsets of D ordered by inclusion form a complete partial order or *domain* D, our domain of descriptions of feature structures.

Deductive closure is used to define the elements of D so that elements defined by equivalent sets of equations are the same. In the rest of this paper, we will specify elements of D by convenient sets of equations, leaving the equations in the closure implicit.

The inclusion order $\sqsubseteq$ in D provides the notion of a description being more or less specific than another. The least-upper-bound operation $\sqcup$ combines two descriptions into the least instantiated description that satisfies the equations in both descriptions, their *unification*. The greatest-lower-bound operation $\sqcap$ gives the most instantiated description containing all the equations common to two descriptions, their *generalization*.

The foregoing definition of consistency may seem very natural, but it has the technical disadvantage that, in general, the union of two consistent sets is not itself a consistent set; therefore, the corresponding operation of unification may not be defined on certain pairs of inputs. Although this does not cause problems at this stage, it fails to deal with the fact that failure to unify is not the same as lack of definition and causes technical difficulties when providing rule denotations. We therefore need a slightly less natural definition.

First we add another statement to the specification of the entailment relation:

- *Falsity.* if e is inconsistent, $\{e\}$ entails every element of D.

That is, falsity entails anything. Next we define Con to be simply the set of all finite subsets of D. The set Con no longer corresponds to sets of equations that are consistent in the usual equational sense.

With the new definitions of Con and $\vdash$, the deductive closure of a set containing an inconsistent equation is the whole of D. The partial order D is now a lattice with top element $\top = D$, and the unification operation $\sqcup$ is always defined and returns $\top$ on unification failure.

We can now define the *description mapping* $\delta : D \to F$ that relates descriptions to the described feature structures. The idea is that, in proceeding from a description $d \in D$ to a feature structure $f \in F$, we keep only definite information about values and discard information that only states value constraints, but does not specify the values themselves. More precisely, seeing d as a set of equations, we consider only the subset $\lfloor d \rfloor$ of d with elements of the form

$$\langle l_1 \cdots l_m \rangle = c_k \quad .$$

Each $e \in \lfloor d \rfloor$ defines an element $f(e)$ of F by the equations

$$\begin{aligned} f(e)(l_1) &= f_1 \\ &\cdots \\ f_{i-1}(l_i) &= f_i \\ &\cdots \\ f_{m-1}(l_m) &= c_k \quad , \end{aligned}$$

with each of the f_i undefined for all other labels. Then, we can define $\delta(d)$ as

$$\delta(d) = \bigsqcup_{e \in \lfloor d \rfloor} f(e) \quad .$$

This description mapping can be shown to be continuous in the sense of domain theory, that is, it has the properties that increasing information in a description leads to nondecreasing information in the described structures (*monotonicity*) and that if a sequence of descriptions approximates another description, the same condition holds for the described structures.

Note that δ may map several elements of D on to one element of F. For example, the elements given by the two sets of equations

$$\left\{ \begin{array}{lcl} \langle f\ h \rangle & = & \langle g\ i \rangle \\ \langle f\ h \rangle & = & c \end{array} \right\} \quad \left\{ \begin{array}{lcl} \langle f\ h \rangle & = & c \\ \langle g\ i \rangle & = & c \end{array} \right\}$$

describe the same structure, because the description mapping ignores the link between $\langle f\ h \rangle$ and $\langle g\ i \rangle$ in the first description. Such links are useful only when unifying with further descriptive elements, not in the completed feature structure, which merely provides feature-value assignments.

Informally, we can think of elements of D as directed rooted graphs and of elements of F as their unfoldings as trees, the unfolding being given by the mapping δ. It is worth noting that if a description is *cyclic*—that is, if it has cycles when viewed as a directed graph—then the resulting feature tree will be infinite.[3]

Stated more precisely, an element f of a domain is *finite*, if for any ascending sequence $\{d_i\}$ such that $f \sqsubseteq \bigsqcup_i d_i$, there is an i such that $f \sqsubseteq d_i$. Then the cyclic elements of D are those finite elements that are mapped by δ into nonfinite elements of F.

5. Providing a Denotation for a Grammar

We now move on to the question of how the domain D is used to provide a denotational semantics for a grammar formalism.

We take a simple grammar formalism with rules consisting of a context-free part over a nonterminal vocabulary $\mathcal{N} = \{N_1, \ldots, N_k\}$ and a set of equations over paths in $([0..\infty] \cdot L^*) \cup C$. A sample rule might be

$$\begin{aligned} S \to\ & NP\ VP \\ & \langle 0\ subj \rangle = \langle 1 \rangle \\ & \langle 0\ predicate \rangle = \langle 2 \rangle \\ & \langle 1\ agr \rangle = \langle 2\ agr \rangle \quad . \end{aligned}$$

This is a simplification of the rule format used in the PATR-II formalism [18,17]. The rule can be read as "an S is an NP followed by a VP, where the *subj*ect of the S is the NP, its *predicate* the VP, and the *agr*eement of the NP the same as the *agr*eement of the VP".

More formally, a grammar is a quintuple $G = \langle \mathcal{N}, S, L, C, R \rangle$, where

- $\mathcal{N}$ is a finite, nonempty set of nonterminals $N_1, \ldots, N_k$
- S is the set of strings over some alphabet (a flat domain with an ancillary continuous function concatenation, notated with the symbol $\cdot$).
- R is a set of pairs $r = \langle N_{r_0} \to N_{r_1} \ldots N_{r_m}, E_r \rangle$, where E_r is a set of equations between elements of $([0..m] \cdot L^*) \cup C$.

As with context-free grammars, local ambiguity of a grammar means that in general there are several ways of assembling the same subphrases into phrases. Thus, the semantics of context-free grammars is given in terms of sets of strings. The situation is somewhat more complicated in our sample formalism. The objects specified by the grammar are pairs of a string and a partial description. Because of partiality, the appropriate construction cannot be given in terms of sets of string-description pairs, but rather in terms of the related domain construction of *powerdomains* [14,19,16]. We will use the *Hoare powerdomain* $P = \mathcal{P}_H(S \times D)$ of the domain $S \times D$ of string-description pairs. Each element of P is an approximation of a *transduction relation*, which is an association between strings and their possible descriptions.

We can get a feeling for what the domain P is doing by examining our notion of *lexicon*. A lexicon will be an

[3] More precisely a *rational* tree, that is, a tree with a finite number of distinct subtrees.

element of the domain P^k, associating with each of the k nonterminals N_i, $1 \leq i \leq k$ a transduction relation from the corresponding coordinate of P^k. Thus, for each nonterminal, the lexicon tells us what phrases are under that nonterminal and what possible descriptions each such phrase has. Here is a sample lexicon:

$$NP: \left\{ \begin{array}{l} (\text{"Uther"}, \\ \quad \{\langle agr\ num\rangle = sg, \langle agr\ per\rangle = 3\}) \\ (\text{"many knights"}, \\ \quad \{\langle agr\ num\rangle = pl, \langle agr\ per\rangle = 3\}) \end{array} \right\}$$

$$VP: \left\{ \begin{array}{l} (\text{"storms Cornwall"}, \\ \quad \{\langle agr\ num\rangle = sg\}) \\ (\text{"sit at the Round Table"}, \\ \quad \{\langle agr\ num\rangle = pl\}) \end{array} \right\}$$

$$S: \quad \{\}$$

By decomposing the effect of a rule into appropriate steps, we can associate with each rule r a denotation

$$[\![r]\!] : P^k \longrightarrow P^k$$

that combines string-description pairs by concatenation and unification to build new string-description pairs for the nonterminal on the left-hand side of the rule, leaving all other nonterminals untouched. By taking the union of the denotations of the rules in a grammar, (which is a well-defined and continuous powerdomain operation,) we get a mapping

$$T_G(\ell) \stackrel{\text{def}}{=} \bigcup_{r \in R} [\![r]\!](\ell)$$

from P^k to P^k that represents a one-step application of all the rules of G "in parallel."

We can now provide a denotation for the entire grammar as a mapping that completes a lexicon with all the derived phrases and their descriptions. The denotation of a grammar is the function that maps each lexicon ℓ into the smallest fixed point of T_G containing ℓ. The fixed point is defined by

$$[\![G]\!](\ell) = \bigsqcup_{i=0}^{\infty} T_G^i(\ell) \quad ,$$

as T_G is continuous.

It remains to describe the decomposition of a rule's effect into elementary steps. The main technicality to keep in mind is that rules state constraints among several descriptions (associated with the parent and each child), whereas a set of equations in D constrains but a single description. This mismatch is solved by embedding the tuple $\langle d_0, \ldots, d_m\rangle$ of descriptions in a single larger description, as expressed by

$$\langle i\rangle = d_i, \qquad 0 \leq i \leq m$$

and only then applying the rule constraints—now viewed as constraining parts of a single description. This is done by the *indexing* and *combination* steps described below. The rest of the work of applying a rule, extracting the result, is done by the *projection* and *deindexing* steps.

The four steps for applying a rule

$$r = \langle N_{r_0} \rightarrow N_{r_1} \ldots N_{r_m}, E_r\rangle$$

to string-description pairs $\langle s_1, d_1\rangle, \ldots, \langle s_k, d_k\rangle$ are as follows. First, we index each d_{r_j} into $d_{r_j}^j$ by replacing every path p in any of its equations with the path $j \cdot p$. We then combine these indexed descriptions with the rule by unifying the deductive closure of E_r with all the indexed descriptions:

$$d = \overline{E_r} \sqcup \bigsqcup_{j=1}^{m} d_{i_j}^j \quad .$$

We can now project d by removing from it all equations with paths that do not start with 0. It is clearly evident that the result d^0 is still deductively closed. Finally, d^0 is deindexed into d_{i_0} by removing 0 from the front of all paths $0 \cdot p$ in its equations. The pair associated with N_{r_0} is then $\langle s_{r_1} \cdots s_{r_m}, d_{r_0}\rangle$.

It is not difficult to show that the above operations can be lifted into operations over elements of P^k that leave untouched the coordinates not mentioned in the rule and that the lifted operations are continuous mappings. With a slight abuse of notation, we can summarize the foregoing discussion with the equation

$$[\![r]\!] = \mathit{deindex} \circ \mathit{project} \circ \mathit{combine}_r \circ \mathit{index}_r$$

In the case of the sample lexicon and one rule grammar presented earlier, $[\![G]\!](\ell)$ would be

$$NP: \quad \{\cdots \text{as before} \cdots\}$$

$$VP: \quad \{\cdots \text{as before} \cdots\}$$

$$S: \left\{ \begin{array}{l} (\text{"Uther storms Cornwall"}, \\ \quad \{\langle subj\ agr\ num\rangle = sg, \ldots\}) \\ (\text{"many knights sit at the Round Table"}, \\ \quad \{\langle subj\ agr\ num\rangle = pl, \ldots\}) \\ (\text{"many knights storms Cornwall"}, \top) \\ \cdots \end{array} \right\}$$

6. Applications

We have used the techniques discussed here to analyze the feature systems of GPSG [15], LFG [2] and PATR-II [17]. All of them turn out to be specializations of our domain D of descriptions. Figure 1 provides a summary of two of the most critical formal properties of context-free-based grammar formalisms, the domains of their feature systems (full F, finite elements of F, or elements of F based on nonrecursive domain equations) and whether the context-free skeletons of grammars are constrained to be *off-line parseable* [13] thereby guaranteeing decidability.

	DCG-II[a]	PATR-II	LFG	GPSG[b]
FEATURE SYSTEM	full	finite	finite	nonrec.
CF SKELETON	full	full	off-line	full

[a]DCGs based on Prolog-II which allows cyclic terms.

[b]HPSG, the current Hewlett-Packard implementation derived from GPSG, would come more accurately under the PATR-II classification.

Figure 1: Summary of Grammar System Properties

Though notational differences and some grammatical devices are glossed over here, the comparison is useful as a *first step* in unifying the various formalisms under one semantic umbrella. Furthermore, this analysis elicits the need to distinguish carefully between the domain of feature structures F and that of descriptions. This distinction is not clear in the published accounts of GPSG and LFG, which imprecision is responsible for a number of uncertainties in the interpretation of operators and conventions in those formalisms.

In addition to formal insights, linguistic insights have also been gleaned from this work. First of all, we note that while the systems make crucial use of unification, generalization is also a well-defined notion therein and might indeed be quite useful. In fact, it was this availability of the generalization operation that suggested a simplified account of coordination facts in English now being used in GPSG [15] and in an extension of PATR-II [8]. Though the issues of coordination and agreement are discussed in greater detail in these two works, we present here a simplified view of the use of generalization in a GPSG coordination analysis.

Circa 1982 GPSG [6] analyzed coordination by using a special principle, the conjunct realization principle (CRP), to achieve partial instantiation of head features (including agreement) on the parent category. This principle, together with the head feature convention (HFC) and control agreement principle (CAP), guaranteed agreement between the head noun of a subject and the head verb of a predicate in English sentences. The HFC, in particular, can be stated in our notation as $\langle 0\ head\rangle = \langle n\ head\rangle$ for n the head of 0.

A more recent analysis [4,15] replaced the conjunct realization principle with a modified head feature convention that required a head to be more instantiated than the parent, that is: $\langle 0\ head\rangle \sqsubseteq \langle n\ head\rangle$ for all constituents n which are heads of 0. Making coordinates heads of their parent achieved the effect of the CRP. Unfortunately, since the HFC no longer forced identity of agreement, a new principle—the nominal completeness principle (NCP), which required that NP's be fully instantiated—was required to guarantee that the appropriate agreements were maintained.

Making use of the order structure of the domains we have just built, we can achieve straightforwardly the effect of the CRP and the old HFC without any notion of the NCP. Our final version of the HFC merely requires that the parent's head features be the *generalization* of the head features of the head children. Formally, we have:

$$\langle 0\ head\rangle = \bigsqcap_{i \in \text{heads of } 0} \langle i\ head\rangle \quad .$$

In the case of parents with one head child, this final HFC reduces to the old HFC requiring identity; it reduces to the newer one, however, in cases (like coordinate structures) where there are several head constituents.

Furthermore, by utilizing an order structure on the domain of constants C, it may be possible to model that troublesome coordination phenomenon, number agreement in coordinated noun phrases [8,15].

7. Conclusion

We have approached the problem of analyzing the meaning of grammar formalisms by applying the techniques of denotational semantics taken from work on the semantics of computer languages. This has enabled us to

- account rigorously for intrinsically partial descriptions,
- derive directly notions of unification, instantiation and generalization,
- relate feature systems in linguistics with type systems in computer science,
- show that feature systems in GPSG, LFG and PATR-II are special cases of a single construction,
- give semantics to a variety of mechanisms in grammar formalisms, and
- introduce operations for modeling linguistic phenomena that have not previously been considered.

We plan to develop the approach further to give accounts of negative and disjunctive constraints [8], besides the simple equational constraints discussed here.

On the basis of these insights alone, it should be clear that the view of grammar formalisms as programming languages offers considerable potential for investigation. But, even more importantly, the *linguistic discipline* enforced by a rigorous approach to the design and analysis of grammar formalisms may make possible a hitherto unachievable standard of research in this area.

References

[1] Ait-Kaci, H. "A New Model of Computation Based on a Calculus of Type Subsumption." Dept. of Computer and Information Science, University of Pennsylvania (November 1983).

[2] Bresnan, J. and R. Kaplan. "Lexical-Functional Grammar: A Formal System for Grammatical Representation." In J. Bresnan, Ed., *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts (1982), pp. 173-281.

[3] Colmerauer, A. "Metamorphosis Grammars." In L. Bole, Ed., *Natural Language Communication with Computers*, Springer-Verlag, Berlin (1978). First appeared as "Les Grammaires de Métamorphose," Groupe d'Intélligence Artificielle, Université de Marseille II (November 1975).

[4] Farkas, D., D.P. Flickinger, G. Gazdar, W.A. Ladusaw, A. Ojeda, J. Pinkham, G.K. Pullum, and P. Sells. "Some Revisions to the Theory of Features and Feature Instantiation." Unpublished manuscript (August 1983).

[5] Gazdar, Gerald and G. Pullum. "Generalized Phrase Structure Grammar: A Theoretical Synopsis." Indiana University Linguistics Club, Bloomington, Indiana (1982).

[6] Gazdar, G., E. Klein, G.K. Pullum, and I.A. Sag. "Coordinate Structure and Unbounded Dependencies." In M. Barlow, D. P. Flickinger and I. A. Sag, eds., *Developments in Generalized Phrase Structure Grammar.* Indiana University Linguistics Club, Bloomington, Indiana (1982).

[7] Harrison, M. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Massachussets (1978).

[8] Kartunnen, Lauri. "Features and Values." *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California (4-7 July, 1984).

[9] Kay, M. "Functional Grammar." *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistic Society*, Berkeley Linguistic Society, Berkeley, California (February 17-19, 1979), pp. 142-158.

[10] Marcus, M., D. Hindle and M. Fleck. "D-Theory: Talking about Talking about Trees." *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Boston, Massachusetts (15-17 June, 1982).

[11] Pereira, F. "Extraposition Grammars." *American Journal of Computational Linguistics 7*, 4 (October-December 1981), 243-256.

[12] Pereira, F. and D. H. D. Warren. "Definite Clause Grammars for Language Analysis—a Survey of the Formalism and a Comparison with Augmented Transition Networks." *Artificial Intelligence 13* (1980), 231-278.

[13] Pereira, F. C. N., and David H. D. Warren "Parsing as Deduction." *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Boston, Massachusetts, (15-17 June, 1983), pp. 137-144.

[14] Plotkin, G. "A Powerdomain Construction." *SIAM Journal of Computing 5* (1976), 452-487.

[15] Sag, I., G. Gazdar, T. Wasow and S. Weisler. "Coordination and How to Distinguish Categories." Report No. CSLI-84-3, Center for the Study of Language and Information, Stanford University, Stanford, California (June, 1982).

[16] Scott, D. "Domains for Denotational Semantics." In *ICALP 82*, Springer-Verlag, Heidelberg (1982).

[17] Shieber, Stuart. "The Design of a Computer Language for Linguistic Information." *Proceedings of the Tenth International Conference on Computational Linguistics* (4-7 July, 1984)

[18] Shieber, S., H. Uszkoreit, F. Pereira, J. Robinson and M. Tyson. "The Formalism and Implementation of PATR-II." In *Research on Interactive Acquisition and Use of Knowledge*, SRI Final Report 1894. SRI International, Menlo Park, California (1983).

[19] Smyth, M. "Power Domains." *Journal of Computer and System Sciences 16* (1978), 23-36.

[20] Stoy, J. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, Massachusetts (1977).

[21] van Emden, M. and R. A. Kowalski. "The Semantics of Predicate Logic as a Programming Language." *Journal of the ACM 23*, 4 (October 1976), 733-742.

[22] Woods, W. et al. "Speech Understanding Systems: Final Report." BBN Report 3438, Bolt Beranek and Newman, Cambridge, Massachusetts (1976).